WHAT IS CLAIMED IS:

1. A method for scaling a source image to produce a destination image, said method comprising the steps of:

calculating a local context metric from a local portion of the source image;

generating a convolution kernel from a plurality of available convolution kernels based on the calculated local context metric; and

using the generated convolution kernel to generate at least one pixel in the destination image.

2. The method as defined in claim 1, further comprising the step of repeating the calculating, generating, and using steps for each pixel in the destination image.

3. The method as defined in claim 1, further comprising the step of:

storing all available convolution kernels in a memory,

wherein in the generating step, one of the stored convolution kernels is selected based on the calculated local context metric.

4. The method as defined in claim 1, further comprising the step of:

storing at least two convolution kernels in a memory,

wherein in the generating step, either one of the stored convolution kernels is selected or another convolution kernel is generated by interpolating the stored convolution kernels.

5. The method as defined in claim 1, wherein the available convolution kernels include at least one smoothing kernel and at least one sharpening kernel.

6. The method as defined in claim 1, wherein the local context metric has more than two possible values.

7. The method as defined in claim 6, wherein the available convolution kernels include a complete smoothing kernel, a complete sharpening kernel, and a plurality of other kernels that provide a transition between the complete sharpening kernel and the complete smoothing kernel.

sub A2

8. A machine-readable medium encoded with a program for scaling a source image to produce a destination image, said program containing instructions for performing the steps of:

calculating a local context metric from a local portion of the source image;

generating a convolution kernel from a plurality of available convolution kernels based on the calculated local context metric; and

using the generated convolution kernel to generate at least one pixel in the destination image.

9. The machine-readable medium as defined in claim 8, wherein said program further contains instructions for performing the step of repeating the calculating, generating, and using steps for each pixel in the destination image.

10. The machine-readable medium as defined in claim 8, wherein said program further contains instructions for performing the step of:

storing all available convolution kernels in a memory,

wherein in the generating step, one of the stored convolution kernels is selected based on the calculated local context metric.

11. The machine-readable medium as defined in claim 8, wherein said program further contains instructions for performing the step of:

storing at least two convolution kernels in a memory,

wherein in the generating step, either one of the stored convolution kernels is selected or another convolution kernel is generated by interpolating the stored convolution kernels.

12. The machine-readable medium as defined in claim 8, wherein the local context metric has more than two possible values.

13. The machine-readable medium as defined in claim 12, wherein the available convolution kernels include a complete smoothing kernel, a complete sharpening kernel, and a plurality of other kernels that provide a transition between the complete sharpening kernel and the complete smoothing kernel.

14. An image scaling device that receives pixels of a source image and outputs pixels of a scaled destination image, said image scaling device comprising:

a context sensor for calculating a local context metric based on local source image pixels;

a kernel generator coupled to the context sensor, the kernel generator generating a current convolution kernel from a plurality of available convolution kernels based on the local context metric calculated by the context sensor; and

a scaler coupled to the kernel generator, the scaler receiving the coefficients of the current convolution kernel from the kernel generator, and using the coefficients to generate at least one pixel of the destination image from pixels of the source image.

15. The image scaling device as defined in claim 14, wherein the context sensor calculates a local context metric for each pixel in the destination image.

16. The image scaling device as defined in claim 14,
wherein the kernel generator stores all available convolution kernels, and
the kernel generator selects one of the stored convolution kernels as the current convolution kernel based on the calculated local context metric.

17. The image scaling device as defined in claim 14,
wherein the kernel generator stores at least two convolution kernels, and
the kernel generator generates the current convolution kernel by either selecting one of the stored convolution kernels or generating another convolution kernel by interpolating the stored convolution kernels.

18. The image scaling device as defined in claim 14, wherein the local context metric has more than two possible values.

19. The image scaling device as defined in claim 18, wherein the available convolution kernels include a complete smoothing kernel, a complete sharpening kernel, and a plurality of other kernels that provide a transition between the complete sharpening kernel and the complete smoothing kernel.

20. A display device that receives source image pixels and displays a scaled destination image, said display device comprising:

a context sensor for calculating a local context metric based on local source image pixels;

a kernel generator coupled to the context sensor, the kernel generator generating a current convolution kernel from a plurality of available convolution kernels based on the local context metric calculated by the context sensor;

a scaler coupled to the kernel generator, the scaler receiving the coefficients of the current convolution kernel from the kernel generator, the scaler using the coefficients to generate at least one pixel of the destination image from pixels of the source image; and

a display for displaying the scaled destination image.

21. The display device as defined in claim 20, wherein the context sensor calculates a local context metric for each pixel in the destination image.

22. The display device as defined in claim 20,

wherein the kernel generator stores all available convolution kernels, and

the kernel generator selects one of the stored convolution kernels as the current convolution kernel based on the calculated local context metric.

23. The display device as defined in claim 20,

wherein the kernel generator stores at least two convolution kernels, and

the kernel generator generates the current convolution kernel by either selecting one of the stored convolution kernels or generating another convolution kernel by interpolating the stored convolution kernels.

24. The display device as defined in claim 20, wherein the display is an LCD display.

ADD
A16

APPENDIX: Algorithm Pseudocode

```
GENERAL FUNCTIONS

   clip(x,min,max)
      x < min : return (min)
      x > max : return (max)
      otherwise : return (x)

   overflow(x,min,max)
      generates an error flag if x < min or x > max

   rs(x,n)
      returns x right-shifted by n positions


Y PHASE GENERATION

   at beginning of frame
   {
      resid = dest_vpix / 2            // start at middle of first pixel
      at new output (destination) line
      {

         if (resid >= dest_vpix)
         {
            if (resid < 2*dest_vpix)   // single step
            {
               resid -= dest_vpix
               index one line ahead in the line buffer
            }
            else                       // double step
            {
               resid -= 2*dest_vpix
               index two lines ahead in the line buffer
            }

         // phase = resid / dest_vpix

         // kill off leading zeroes of dest_pix to create denominator
         // between 16 and 31 and numerator between 0 and 31
         casex(dest_pix[11:4])
            8'b1xxxxxxx:
               begin numer=resid_q[11:7]; denom=dest_pix[11:7]; end
            8'b01xxxxxx:
               begin numer=resid_q[10:6]; denom=dest_pix[10:6]; end
            8'b001xxxxx:
               begin numer=resid_q[9:5]; denom=dest_pix[9:5]; end
            8'b0001xxxx:
               begin numer=resid_q[8:4]; denom=dest_pix[8:4]; end
```

```
            8'b0000lxxx:
               begin numer=resid_q[7:3]; denom=dest_pix[7:3]; end
            8'b000001xx:
               begin numer=resid_q[6:2]; denom=dest_pix[6:2]; end
            8'b0000001x:
               begin numer=resid_q[5:1]; denom=dest_pix[5:1]; end
            8'b00000001:
               begin numer=resid_q[4:0]; denom=dest_pix[4:0]; end
         endcase

         phase = floor (numer/denom)

         resid += src_vpix
         overflow(resid,0,4095)

      }
   }
```

X PHASE GENERATION

```
   at beginning of line
   {
      resid = dest_hpix / 2                // start in middle of pixel
      at new output (destination) pixel    // every dclk
      {

         if (resid >= dest_hpix)
         {
            if (resid < 2*dest_hpix)       // single step
            {
               resid -= dest_hpix
               index one pixel ahead in the scale_reg FIFO
            }
            else                          // double step
            {
               resid -= 2*dest_hpix
               index two pixels ahead in the scale_reg FIFO
            }

         // phase = resid / dest_hpix

         // kill off leading zeroes of dest_pix to create denominator
         // between 16 and 31 and numerator between 0 and denominator-1
         casex(dest_pix[11:4])
            8'blxxxxxxx:
               begin numer=resid_q[11:7]; denom=dest_pix[11:7]; end
            8'b01xxxxxx:
               begin numer=resid_q[10:6]; denom=dest_pix[10:6]; end
            8'b001xxxxx:
               begin numer=resid_q[9:5]; denom=dest_pix[9:5]; end
```

```
                8'b0001xxxx:
                   begin numer=resid_q[8:4]; denom=dest_pix[8:4]; end
                8'b00001xxx:
                   begin numer=resid_q[7:3]; denom=dest_pix[7:3]; end
                8'b000001xx:
                   begin numer=resid_q[6:2]; denom=dest_pix[6:2]; end
                8'b0000001x:
                   begin numer=resid_q[5:1]; denom=dest_pix[5:1]; end
                8'b00000001:
                   begin numer=resid_q[4:0]; denom=dest_pix[4:0]; end
                default:
                   begin numer=resid_q[4:0]; denom=dest_pix[4:0]; end
             endcase

             phase = floor (numer/denom)

             resid += src_hpix
             overflow(resid,0,4095)

          }
       }

   CONTEXT SENSOR

      For the y dimension
         // R0,R1,R2 = red channel (of 3 input lines, 8b unsigned data)
         // G0,G1,G2 = green channel
         // B0,B1,B2 = blue channel

         y_max_r = max(rs(R0,2),rs(R1,2),rs(R2,2))
         y_min_r = min(rs(R0,2),rs(R1,2),rs(R2,2))
         y_delta_r = y_max_r - y_min_r

         // same for y_delta_g, y_delta_b

         y_delta = y_delta_r + y_delta_g + y_delta_b

         if (y_delta < 64)
         {
            y_context = rs(y_delta,2)
         }
         else
         {
            y_context = 15
         }

      For the x dimension
         // R0,R1,R2=red channel intermediate filter results (11b 2's comp)
         // G0,G1,G2=green channel intermediate filter results
         // B0,B1,B2=blue channel intermediate filter results
```

```
      x_max_r = max(rs(R0,5),rs(R1,5),rs(R2,5))
      x_min_r = min(rs(R0,5),rs(R1,5),rs(R2,5))
      x_delta_r = x_max_r - x_min_r

      // same for x_delta_g, x_delta_b

      x_delta = x_delta_r + x_delta_g + x_delta_b

      if(x_delta < 16)
      {
         x_context = x_delta
      }
      else
      {
         x_context = 15
      }
```

FILTER

```
   // variables
      y_kernel[256][3] = array of 6b filter coefficients
      x_kernel[256][5] = array of 6b filter coefficients
      in = 8b unsigned input data

   temp = 0;
   for (x=0;x<5;x++)
   {
      y_inter = 0;
      for (y=0;y<3;y++)
      {
         y_product = in * y_kernel[y_context*16 + y_phase][y];
         y_inter += rs(y_product,4);
      }
      x_product = y_inter * x_kernel[x_context*16 + x_phase][y];
      x_inter = rs(x_product,5);
      overflow(x_inter,-1024,1023);     // not all MSBs are carried here
      temp += x_inter;
      overflow(temp,-1024,1023);        // not all MSBs are carried here
   }
   overflow(temp,-1024,1023);           // not all MSBs are carried here
   out = clip(rs(temp,1),0,255);
```